# MIPS CPU 设计报告

重庆大学 预备队
吕昱峰，张启航，葛胡军，蔡林晋

## 一、设计简介

参赛 CPU 设计为哈佛结构的 32 位五级流水线的 CPU Core，实现了预赛要求的 57 条指令；设计实现 4KB 大小，块大小为 1word，处理方式为写直达、直接映射的 I-Cache 与 D-Cache；并结合 Cache 仲裁设计改造对接 AXI 总线的接口。

### （一）设计变更说明

暂无

# 二、设计方案

## （一）总体设计思路

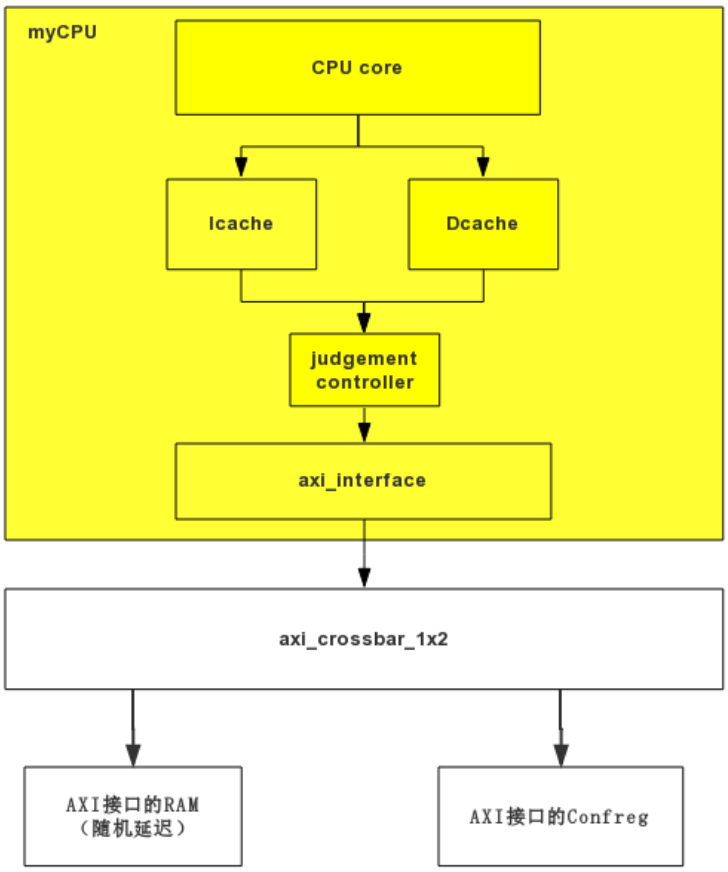


图 1

如图 1，总体设计采用软核与外围分离的思路，MIPS CPU 软核采用哈佛结构，与通用 MIPS CPU 设计思路相同，预留 INST_RAM 与 DATA_RAM 交互接口，可直接与双内存结构的 SRAM 环境对接进行初步功能测试，也可直接连接 Cache，为后续性能加速做准备。

接口模块不使用赛方提供的 AXI 接口，而使用与 Cache 模块结合的设计，对接口进行改造，将双端口改为多端口，仲裁不在接口模块处理，而经 Cache 流出后由单独的总裁控制进行处理，输出仅为单端口，简化接口设计。

其中 MIPS Core 模块设计如图 2 所示，采用组成原理课本的经典五级流水设计图，先实现所有所需器件后，参照此图进行连接，即形成初步的五级流水 Core，针对流水线的冒险处理和异常处理则由 Hazard 与 Exception 模块进行处理。

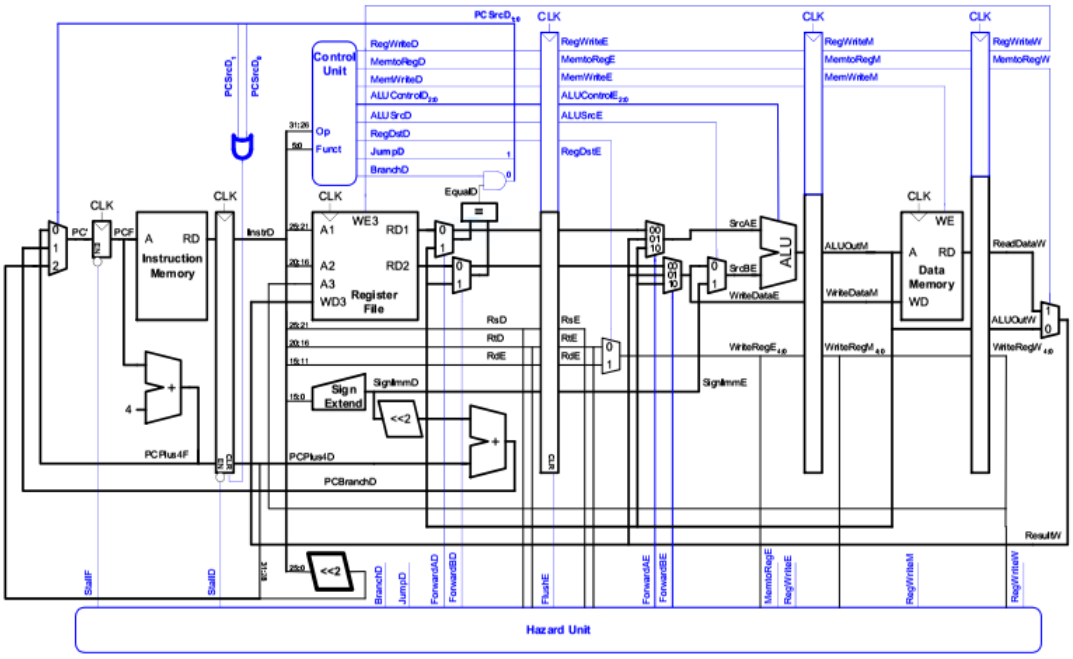


图 2

如图 3，CPU 主要由两大模块组成：Controller 模块和 DataPath 模块。其中，Controller 模块包含图 2 上部蓝色部分，hazard unit 不单独外置，放入 DataPath 中。

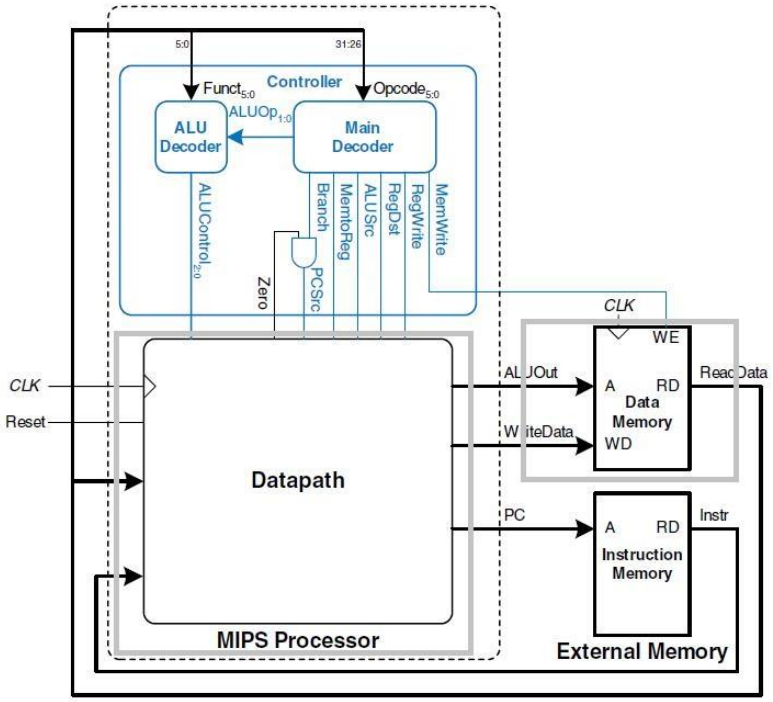


图 3

Controller 模块作用为解析指令，根据指令 inst 的高 6 位 op 和低 6 位 funct 来判断出具体的指令，再根据具体的指令设置数据通路的各个选择的信号，alu 的控制信号和 Hilo 寄存器的控制信号等，通过触发器从译码阶段传到接下来的各个阶段，并根据需要选择清空或暂停，或传到 DataPath 的数据通路中去。

DataPath 模块为五级流水的数据通路，其中有取址，译码，执行，访存，回写五个阶段，各个阶段根据 Controller 传过来的信号实现自己的功能处理不同的指令。此外在该模块中还有协处理器 CP0，控制模块 hazard unit，hilo 寄存器，和异常处理模块 Exception。

## （二）Controller 模块设计

### 1、模块示意图

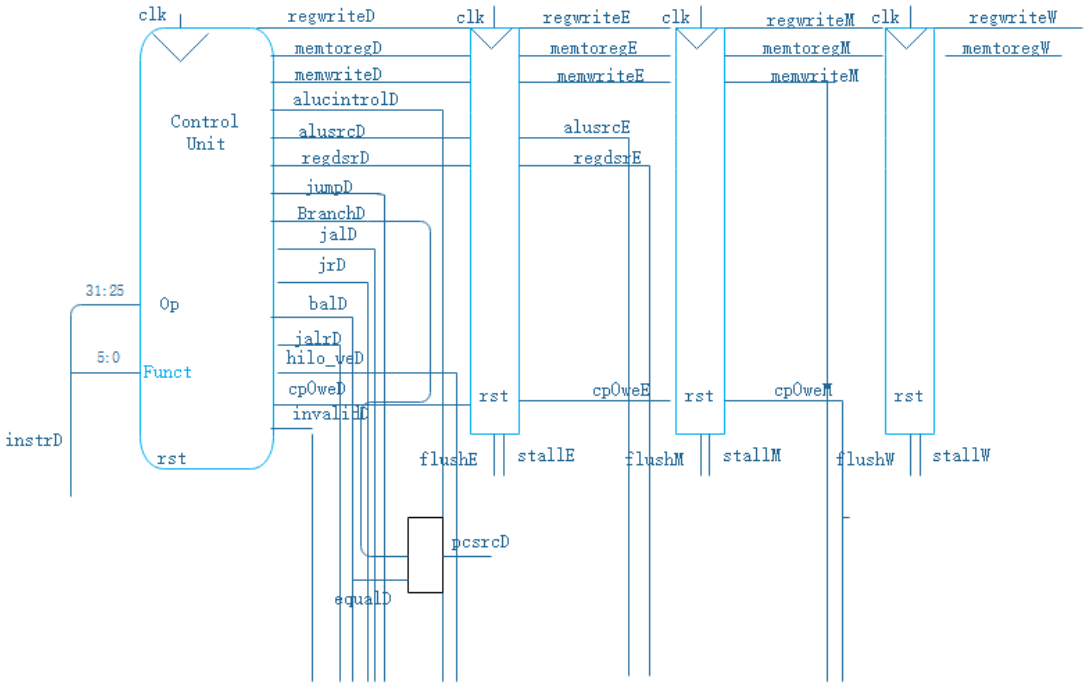


图 4

### 2、端口介绍

表 1

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | rst | 1 | 输入 | 复位信号 |
| 3 | instrD | 32 | 输入 | 输入的指令 |

| | | | | |
|---|---|---|---|---|
| 4 | equalD | 1 | 输入 | 分支指令的条件判断结果 |
| 5 | flushE | 1 | 输入 | 执行阶段触发器清空信号 |
| 6 | stallE | 1 | 输入 | 执行阶段触发器暂停信号 |
| 7 | stallM | 1 | 输入 | 访存阶段触发器暂停信号 |
| 8 | flushM | 1 | 输入 | 访存阶段触发器清空信号 |
| 9 | flushW | 1 | 输入 | 回写阶段触发器清空信号 |
| 10 | PCsrcD | 1 | 输出 | 分支指令在译码阶段的跳转信号 |
| 11 | branchD | 1 | 输出 | 译码阶段判断是否是分支指令 |
| 12 | jumpD | 1 | 输出 | 译码阶段判断是否是跳转指令 J |
| 13 | jalD,jrD, balD,jarD | 1 | 输出 | 译码阶段判断是否是跳转指令 JAL,JR, 和是否为要写回的分支指令 |
| 14 | alucontrolD | 5 | 输出 | 译码阶段控制 alu 做运算的信号 |
| 15 | hilo_weD | 2 | 输出 | 译码阶段 hilo 寄存器的写入控制信号 |
| 16 | invalidD | 1 | 输出 | 译码阶段无效指令的控制信号 |
| 17 | memtoregE | 1 | 输出 | 执行阶段加载指令的输出选择信号 |
| 18 | alusrcE | 1 | 输出 | 执行阶段控制传入 alu 的操作数是否是立即数的扩展 |
| 19 | regdstE | 1 | 输出 | 执行阶段控制寄存器的写入地址 |
| 20 | memtoregM | 1 | 输出 | 访存阶段加载指令的输出选择信号 |
| 21 | memwriteM | 1 | 输出 | 访存阶段数据存储器使能信号 |
| 22 | regwriteE | 1 | 输出 | 执行阶段的指令是否要写回寄存器信号 |
| 23 | regwriteM, | 1 | 输出 | 访存阶段的指令是否要写回寄存器信号 |
| 24 | cp0weM, | 1 | 输出 | CP0 协处理器的使能端的写入使能信号 |
| 25 | memtoregW | 1 | 输出 | 回写阶段加载指令的输出选择信号 |
| 26 | regwriteW | 1 | 输出 | 回写阶段的指令是否要写回寄存器信号 |

## 3、内部的数据通路和控制逻辑

Controller 模块根据传入的指令 inst，通过 maindec 模块解析出指令的类型，给其在数据通路的选择信号赋值，然后将赋值后的信号通过触发器向后传输，当 DataPath 模块中有某阶段用到某信号，Controller 就将该信号传至该阶段，触发器可以根据 DataPath 中的信号选择清空或者暂停。这些控制信号中 PCsrcD 是需要根据 branchD 和 equalD 信号决定，即只有指令是分支指令且分支条件满足才打开。Controller 模块还通过 aludec 模块决定具体的指令需要做何种运算的控制信号 alucontrol。

## 4、maindec (主译码器）

### （1）端口介绍

表 2

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | instr | 32 | 输入 | 译码阶段的指令 |
| 2 | memtoreg | 1 | 输出 | 译码阶段的指令是否要写入存储器中 |
| 3 | branch | 1 | 输出 | 译码阶段的指令是否是分支指令 |
| 4 | alusrc | 1 | 输出 | 控制传入 alu 的操作数是否是立即数的扩展 |
| 5 | regdst | 1 | 输出 | 控制寄存器的写入地址 |
| 6 | regwrite | 1 | 输出 | 译码阶段的指令是否要写回寄存器 |
| 7 | jump | 1 | 输出 | 译码阶段的指令是否是 jump 指令 |
| 8 | aluop | 4 | 输出 | 控制 aludec 做 alu 运算的具体判断 |
| 9 | jal,jr,jal,bal | 1 | 输出 | 译码阶段的指令是否是跳转指令 |
| 10 | hilo_we | 2 | 输出 | 控制 Hilo 模块是否需要写入 |
| 11 | cp0we | 1 | 输出 | 控制 CP0 模块是否需要写入 |
| 12 | invalid | 1 | 输出 | 译码阶段是否是有效的指令 |

### （2）内部的数据通路和控制逻辑

Maindec 模块通过 inst 传入指令，通过高 6 位 op 和低 6 位 funct 判断是什么类型的指令，通过 controls 设置所有信号，通过 aluop 传到在 aludec 里面进行子判断，当没有在指令集的指令被判为无效指令，及 invalid 置为 1.其中 R-type 类型的数据通路一样，所以统一设置，其中对于 op 为 000000 时的特殊的指令再次判断 funct 段来单独设置控制信号。

## 5、Aludec (alu 译码器)

### （1）端口介绍

表 3

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | func | 6 | 输入 | 译码阶段的指令的低六位 |
| 2 | aluop | 4 | 输入 | 译码阶段的指令类型 |
| 3 | alucontrol | 5 | 输出 | 控制 alu 要进行的运算类型 |

### （2）内部的数据通路和控制逻辑

Aludec 模块根据 maidec 模块传入的 aluop 和译码阶段的指令的 funct 来设置传出到 alu 的控制信号 alucontrol。

## （三）DataPath 模块设计

### 1、取址

#### （1）示意图

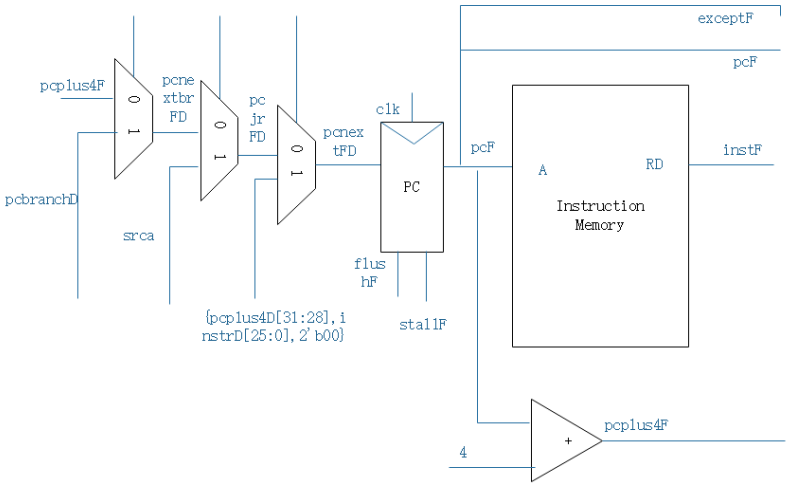


图 5

#### （2）端口介绍

PC 模块端口：

表 4

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | rst | 1 | 输入 | 复位信号 |
| 3 | en | 1 | 输入 | 使能信号 |
| 4 | clr | 1 | 输入 | 清空信号 |
| 5 | d，t | 32 | 输入 | 输入的指令 |
| 7 | q | 32 | 输出 | 输出的指令 |

#### （3） 内部的数据通路和控制逻辑

使用三个个二选一多路选择器，选择下一条指令的 PC 值。选择的 PC 有：

- PCplus4F（当前 PC 加 4 之后的 PC 值），
- PCbranchD（由立即数 offset 左移 2 位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的 PC），

- srca3D（PC 为寄存器 rs 中的值），
- {PCplus4D[31:28],instrD[25:0],2'b00}（分支指令对应的延迟槽指令的 PC 的最高 4 位与立即数 instr_index 左移 2 位后的值拼接得到 PC 值）

选择出的 PC 值进入 PC 模块中根据清空和复位信号得到取址阶段的最终 PC 值。该值作为 DataPath 的一个输出到 INST_RAM 中，INST_RAM 读出 PC 值得到对应的指令再传回 DataPath 的取址阶段作为 instD 的值就是将要处理的指令了，完成取址阶段的任务。

## 2、译码

### （1） 示意图

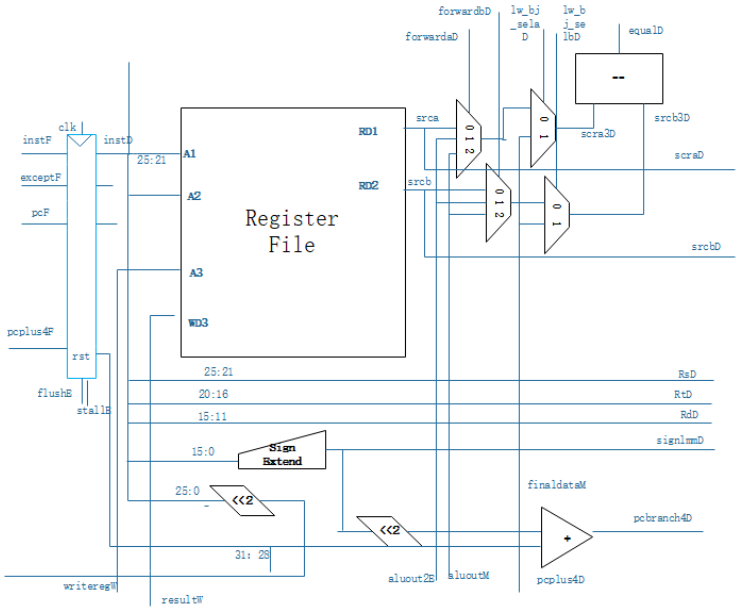


图 6

### （2） 端口介绍

表 5

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| regfile 模块 | | | | |
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | we3 | 1 | 输入 | 使能信号 |
| 3 | ral | 5 | 输入 | 第一个读寄存器端口要读取的寄存器地址 |
| 4 | ra2 | 5 | 输入 | 第二个读寄存器端口要读取的寄存器地址 |

| | | | | |
|---|---|---|---|---|
| 5 | wa3 | 5 | 输入 | 要写入的寄存器地址 |
| 5 | wd3 | 32 | 输入 | 要写入的数据 |
| 5 | rd1 | 32 | 输出 | 第一个读寄存器端口读出的寄存器值 |
| 5 | rd2 | 32 | 输出 | 第二个读寄存器端口读出的寄存器值 |
| sign extend 模块 | | | | |
| 1 | a | 16 | 输入 | 输入的立即数值 |
| 2 | type | 2 | 输入 | 选择信号 |
| 3 | y | 32 | 输出 | 输出的扩展值 |
| eqcmp 模块 | | | | |
| 1 | a | 32 | 输入 | 第一个操作数 |
| 2 | b | 32 | 输入 | 第二个操作数 |
| 3 | op | 6 | 输入 | 指令的高六位 |
| 4 | rt | 5 | 输入 | 指令的 16 到 20 位 |
| 5 | y | 1 | 输出 | 判断的结果 |

### （3）内部的数据通路和控制逻辑

取址阶段主要的器件是 regfile 寄存器，regfile 寄存器有 32 个 32 位寄存器的，若写使能端为 0，则两个读寄存器端口读出的就是俩个读寄存器端口的读取的地址里面的值。若写使能端打开，这根据写入的寄存器的地址和写入的数据到寄存器中。Regfile 的输出将传到执行阶段作为 alu 的俩个操作数，还传到 equalD 模块作为分支指令的两个比较值。

Sign extend 模块是给传入的立即数即 inst 的 0 到 15 位扩展。将 inst 的 29 和 28 作为选择信号，只有在 11 的时候做无符号扩展。

EquLD 模块根据传入的 inst 的高六位的值 op 和 inst20 到 16 位 rt 做具体判断是何种分支指令，具体比较传入的操作数。

## 3、执行（除法模块）

DataPath 的执行阶段主要包含了：多个用于保存 D 阶段状态的触发器、一个 ALU 模块、一个除法模块以及多个多路选择器。ALU 的接口如下：

表 6

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | a | 32 | 输入 | 操作数 1 |
| 2 | b | 32 | 输入 | 操作数 2 |
| 3 | sa | 5 | 输入 | 移动位数（对 shift 指令） |
| 4 | alucontrol | 5 | 输入 | alu 控制信号 |
| 5 | hi_in | 32 | 输入 | hi 寄存器读取结果 |
| 6 | lo_in | 32 | 输入 | lo 寄存器读取结果 |
| 7 | cp0data | 32 | 输入 | CP0 读取结果 |

| | | | | |
|---|---|---|---|---|
| 8 | y | 32 | 输出 | alu 运算结果 |
| 9 | overflow | 1 | 输出 | 溢出标志 |
| 10 | hi_alu_out | 32 | 输出 | 待写入 hi 寄存器的值（对乘法指令） |
| 11 | lo_alu_out | 32 | 输出 | 待写入 lo 寄存器的值（对乘法指令） |

alu 根据控制信号（alucontrol）对操作数执行相应操作。对于普通指令，如 R 型指令、移位指令等，alu 对操作数 a、b 进行运算，将结果输出到 y 端口。对于移位指令，移动位数由输入 sa 给出，移动方向以及是算术移位还是逻辑移位由 alucontrol 控制。对于乘法指令，结果的高低位分别通过 hi_alu_out、lo_alu_out 端口写到 hilo 寄存器中（在回写阶段）。对有符号运算，溢出时 overflow 置 1.

对于对 hilo 寄存器进行存取操作的指令，采取复用 alu 通路的策略，因此 alu 接受两个从 hilo 寄存器读出的结果：hi_in 和 lo_in，对于这 mfhi、mflo 指令，alu 仅将 hi_in、lo_in 复制到 y；对于 mthi、mtlo 指令，alu 将 hi_alu_out、lo_alu_out 赋为 a，a 为需要复制进 hi/lo 寄存器的值，为 rs 的值。对于需要对 CP0 中寄存器进行存取的指令，处理方法类似。对 mfc0 指令，y 被赋为 cp0data；对 mtc0，y 被赋值为 b，b 为待写入 CP0 中寄存器的值，为 rt 段，其在通过 alu 后，在 M 阶段写入相应寄存器中。

另一个与 alu 并列模块是除法器（divider_Primary），除法器的端口如下：

表 7

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | rst | 1 | 输入 | 重置信号 |
| 3 | op | 5 | 输入 | 控制信号（无符号还是有符号除法） |
| 4 | opdata1_i | 32 | 输入 | 被除数输入 |
| 5 | opdata2_i | 32 | 输入 | 除数输入 |
| 6 | annul_i | 1 | 输入 | 除法器使能信号，低电平有效 |
| 7 | result_o | 64 | 输出 | 运算结果 |
| 8 | ready_o | 1 | 输出 | 运算结束标志，运算完成时置为 1 |
| 9 | start_i | 1 | 输入 | 开始信号，置 1 时开始运算 |

除法模块的两个操作数输入 opdata1_i、opdata2_i 与 alu 的输入 a、b 连接到相同位置，其控制信号也与 alu 的控制信号连接到相同位置。除法器的 start_i 与 ready_o 都与 hazard 模块连接。当 hazard 将 start_i 置为 1，除法开始运算，hazard 将流水线暂停，18 个周期过后得到运算结果，ready_o 置为 1 并输出到 hazard，hazard 取消暂停流水线。

执行阶段涉及的多路选择器主要用于实现对 alu 运算数输入的选择以及用于实现数据前推。为了设计的可扩展性，我们倾向于使用三选一多路选择器和二选一多路选择器，采用多个多路选择器层叠的方式进行信号选择。选择信号由 Controller 模块和 hazard 模块给出。

Controller 选择输入 alu 的操作数，如：是 rt 还是扩展后的立即数。

## 4、访存

DataPath 的访存阶段主要包含了多个存储 M 阶段状态的触发器，以及一个数据存储器（用的 Xilinx 的 IP 核）。触发器中的值由 E 阶段向后传播得到。为了实现按半位和按字节存储，我们给数据存储器模块加上了一个外壳：memsel 模块，该模块的端口如下：

表 8

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | PC | 32 | 输入 | 程序计数器的值 |
| 2 | op | 6 | 输入 | 指令的 op 段 |
| 3 | addr | 32 | 输入 | 地址 |
| 4 | writedata | 32 | 输入 | 待写入数据 |
| 5 | readdata | 32 | 输入 | 读出的数据 |
| 6 | sel | 4 | 输出 | 选择信号，以实现按字节、半字读写 |
| 7 | writedata2 | 32 | 输出 | 输入到数据存储器的数据 |
| 8 | finaldata | 32 | 输出 | 从数据存储器读出的数据 |
| 9 | bad_addr | 32 | 输出 | 非法地址 |
| 10 | adelM | 1 | 输出 | 读取指令标志位 |
| 11 | adesM | 1 | 输出 | 保存指令标志位 |

memsel 模块根据 op 判定是哪种存取指令，根据指令类型对 sel、adelM、adesM 进行赋值，并得到传到数据存储器的值 writedata2。如果是读取指令，adelM 置为 1，若为存储指令，adesM 置为 1。

memsel 模块对从数据存储器读出的值进行加工，得到 finaldata。sel 可以理解为写入数据存储器的使能信号，如 sel 为 1100 表示对高半位执行写入，为 0001 表示对低字节执行写入。如果出现不是按字操作且地址最后两位不为 0 的情况，bad_addr 为输入 memsel 的地址，否则为 PC 的值。

memsel 与数据存储器结合，可以实现对数据的按字节、按半字读取，即实现 LW、LB、SB 等指令。

## 5、回写（Hilo 寄存器）

DataPath 的回写阶段主要包含相应的用于储存 W 阶段状态的触发器，以及用以选择写入数据的多路选择器。触发器中的值由 M 阶段向后传播得到。写入的数据可以是 alu 的运算结果、从数据存储器中读出的值等等。hi、lo 寄存器的回写也是在这个阶段完成，hi、lo 寄存器的端口如下：

表 9

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | rst | 1 | 输入 | 重置信号 |
| 3 | we | 2 | 输入 | 写入使能 |
| 4 | hi | 32 | 输出 | 写入 hi 寄存器的值 |
| 5 | lo | 32 | 输出 | 写入 lo 寄存器的值 |
| 6 | hi_o | 1 | 输出 | 从 hi 寄存器读出的值 |
| 7 | lo_o | 64 | 输出 | 从 lo 寄存器读出的值 |

## 6、Exception

### （1）模块的功能意图

用来判断异常类型，将前四个阶段的异常信号与中断异常综合成一个总输出信号，将其传入 CP0 模块中为异常处理做准备。

### （2）端口介绍

表 10

| 序号 | 接口名 | 宽度(bit) | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | except | 8 | 输入 | 记录前三个阶段是否出现异常 |
| 3 | adel | 1 | 输入 | 加载指令地址是否有错 |
| 4 | ades | 1 | 输入 | 写入指令地址是否有错 |
| 5 | cp0_status | 32 | 输入 | 用于中断异常判断 |
| 6 | cp0_cause | 32 | 输入 | 用于中断异常判断 |
| 7 | excepttype | 32 | 输出 | 综合后的异常信号 |

### （3）内部的数据通路和控制逻辑

Except 信号从取指阶段一直传到存储阶段，记录了地址错例外——第一阶段，非法指令，内陷指令，ERET 指令——第二阶段，整型溢出例外——第三阶段，而地址错例外——访存阶段的判断由 memsel 模块中的 adel 与 ades 传入 Exception 模块，中断则是由 CP0 中的 cause 与 status 寄存器的值传入 Exception 进行判断。这样，所有的异常类型都得到了判断，最终输出综合的异常信号 Excepttype，并将其传入 CP0 模块中进行异常处理。

## 7、CP0

### （1）模块的功能意图

CP0 为协处理器，主要负责配置 CPU 的工作状态、高速缓存控制、异常控制等工作。CP0 实现了其中的 Count,Compare,Status 等八个寄存器来对出现异常时的情况进行处理以及写入或读出，出现异常的类型由的 Exception 模块进行判断并传入 CP0 寄存器中。

（2）端口介绍

表 11

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | rst | 1 | 输入 | 复位信号 |
| 3 | we_i | 1 | 输入 | CP0 写使能信号 |
| 4 | waddr_i | 5 | 输入 | 要写入的 CP0 中寄存器的地址 |
| 5 | raddr_i | 5 | 输入 | 要读取的 CP0 中寄存器的地址 |
| 6 | data_i | 32 | 输入 | 要写入 CP0 中寄存器的数据 |
| 7 | int_i | 6 | 输入 | 6 个外部硬件中断输入 |
| 8 | excepttype_i | 32 | 输入 | 异常类型信号 |
| 9 | current_inst_addr_i | 32 | 输入 | 当前指令地址 |
| 10 | is_in_delayslot_i | 1 | 输入 | 指令是否在延迟槽中 |
| 11 | bad_addr_i | 32 | 输入 | 判断出的非法地址 |
| 12 | data_o | 32 | 输出 | 读出的 CP0 中寄存器的值 |
| 13 | count_o | 32 | 输出 | Count 寄存器的值 |
| 14 | compare_o | 32 | 输出 | Compare 寄存器的值 |
| 15 | status_o | 32 | 输出 | Status 寄存器的值 |
| 16 | cause_o | 32 | 输出 | Cause 寄存器的值 |
| 17 | ePC_o | 32 | 输出 | EPC 寄存器的值 |
| 18 | config_o | 32 | 输出 | Config 寄存器的值 |
| 19 | prid_o | 32 | 输出 | PRID 寄存器的值 |
| 20 | badvaddr | 32 | 输出 | 非法地址 |
| 21 | timer_int_o | 1 | 输出 | 是否有定时中断发生 |

（3）内部的数据通路和控制逻辑

CP0 模块中既可以根据输入的异常信号，对内部的寄存器数据进行修改与读写，又可以单纯依据 MFC0 与 MTC0 指令作为寄存器进行正常的读写操作。

如果为 MFC0 与 MTC0 指令，其实现方式和 MFHI/MTHI 指令实现方式一样，只是单纯的读写值，若为 MTC0 指令，则打开使能信号，同时由 waddrr_i 信号判断要写入的地址，对寄存器进行相应的修改。为 MFC0 指令时，由 data_o 对想要读取的寄存器的值进行读出。

如果输入的 excepttype_i 信号不为 0(即有异常发生)，则会根据异常种类以及该指令是否在延迟槽中对 EPC,Cause,Status 等寄存器的值进行修改，为接下来的异常处理做好准备工

作。

## 8、Hazard

### （1）模块的功能意图

冒险模块，处理所有数据冒险，控制冒险的情况，进行数据前推，流水线暂停与清零，进入异常处理入口等工作。

### （2）端口介绍

由于 hazard 模块牵扯信号过多，需要进行大量判断，在此只选择一部分重要端口进行介绍。

表 12

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | stall(四阶段) | 1 | 输出 | 流水线暂停 |
| 2 | flush（五阶段） | 1 | 输出 | 流水线清空 |
| 3 | lw_bj_selD | 1 | 输出 | 判断加载后是否跟跳转 |
| 4 | invalidD | 1 | 输入 | 无效指令 |
| 5 | alucontrolE | 5 | 输入 | Alu 控制信号 |
| 6 | regwriteE | 1 | 输入 | 回写信号 |
| 7 | writereg2E | 5 | 输入 | 回写寄存器地址 |
| 8 | forwarda(统称) | 1 | 输出 | 处理 rs 寄存器数据前推 |
| 9 | forwardb（统称） | 1 | 输出 | 处理 rd 寄存器数据前推 |
| 10 | div_start | 1 | 输入 | 除法开始信号 |
| 11 | div_ready | 1 | 输入 | 除法结束信号 |
| 12 | cp0weM | 1 | 输入 | Cp0 写使能信号 |
| 13 | newPCM | 32 | 输入 | 新的 PC 值 |
| 14 | excepttypeM | 32 | 输入 | 异常信号 |

### （3）内部的数据通路和控制逻辑

Hazard 基本分为三个部分，第一部分为流水线的暂停和清零，第二部分为数据前推，第三部分为异常信号的处理。

对于流水线的暂停，考虑 LW 类指令的暂停，除法指令的暂停，以及出现异常与无效指令时的暂停。对不同阶段的暂停做了不同阶段的考虑。

对于流水线的清空，除了执行阶段还要考虑是否为 LW 指令外，其他清空的原因都是因为要进行异常处理。

数据前推，是一个涉及阶段广，涉及指令多的问题。综合考虑了译码与执行两个阶段，

逻辑运算指令、分支跳转指令、读写 hilo 指令、读写 cp0 指令四种类型指令。实现了总共四种类型的前推。对上条指令写入的寄存器与本条指令要读的寄存器进行了综合判断，进行了合理的数据前推。

若判断存在异常信号，将 PC 值置为 bfc00380，使得下一个周期进入自带的异常处理模块，否则正常执行。

## （四）Cache 模块设计

### 1、模块示意图

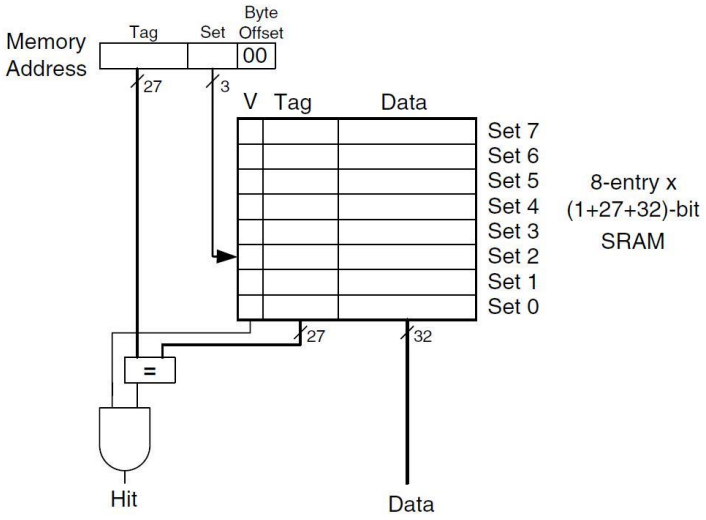


图 7

### 2、端口介绍

表 13

| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
|---|---|---|---|---|
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | clrn | 1 | 输入 | 复位信号 |
| 3 | p_a | 32 | 输入 | CPU 输入到 Cache 的地址 |
| 4 | p_dout | 32 | 输入 | CPU 输入到 Cache 的值 |
| 5 | p_din | 32 | 输出 | Cache 输出到 CPU 的值 |
| 6 | p_strobe | 1 | 输入 | CPU 输入的访存使能信号 |
| 7 | p_wen | 4 | 输入 | CPU 输入的字节片选信号 |
| 8 | p_size | 2 | 输入 | CPU 输入的访问字节数信号 |

| | | | | |
|---|---|---|---|---|
| 9 | p_rw | 1 | 输入 | CPU 输入的读写信号，1 为写，0 为读 |
| 10 | p_ready | 1 | 输出 | Cache 输出的 ready 信号，表示访存完成 |
| 11 | m_a | 32 | 输出 | 输出至内存的地址 |
| 12 | m_dout | 32 | 输入 | 由内存输入 Cache 的值 |
| 13 | m_din | 32 | 输出 | 由 Cache 输出至内存的值 |
| 14 | m_strobe | 1 | 输出 | 内存使能信号 |
| 15 | m_wen | 4 | 输出 | 内存访问字节片选信号 |
| 16 | m_size | 2 | 输出 | 内存访问字节数信号 |
| 17 | m_rw | 1 | 输出 | 内存读写信号，1 为写，0 为读 |
| 18 | m_ready | 1 | 输入 | 内存访存完成信号 |

### 3、Cache 内部设计与仲裁机制

如图 7 所示，Cache 使用 reg 进行存储，其包含三部分，分别为 DTag、DValid、DData，Cache 内部还包括 Cache 控制部分和 hit 对比部分，当地址的 Tag 与 DTag 相同地址的 Tag 一致且 DValid 相同地址为 1 时，Cache 命中，单周期返回结果，否则对内存进行操作。由于采用写直达方式，每次写操作需对内存写入并对 Cache 写入，读操作时也将内存输出的值存入 Cache 中。

Cache 输出至内存的值由控制部分决定，控制部分为组合逻辑，在未 hit 的情况下，对 strobe、wen、size、rw 进行赋值。

由于接口为单端口，因而需要考虑 ICache 与 DCache 同时对内存进行操作的情况，我们设定 ICache 优先级更高，即遇到同时访存时，优先进行 ICache 操作，DCache 等待，待 ICache 操作完成后，DCache 继续访存。

同时，Cache 对内存进行操作时，由于当前流水线阶段未能完成执行，因而需要输出暂停信号，暂停信号由 p_ready 取反得到，并输入至 hazard 模块，在 hazard 模块中对不用阶段进行暂停处理。

其中，ICache 输出的暂停信号需暂停取指、译码阶段，暂停译码阶段是为了保证延迟槽顺序；DCache 输出的暂停信号需暂停访存阶段及其之前的所有阶段，并清空回写阶段的值。

## （五）AXI 接口模块设计

由于 AXI 接口改为单端口，无需在内部进行仲裁，因而仅需要与 AXI 总线信号进行判断并进行状态转移即可，故而在此不进行详述。

# 三、设计结果

## （一）设计交付物说明

表 14

| | |
|---|---|
| **\|-score.xls** | Excel 表格，包含功能测试、性能测试得分的计算 |
| **\|-design.pdf** | myCPU 设计报告 |
| **\|-soc_axi_func/** | 自实现 CPU 的功能测试环境，AXI 接口 |
| **\| \|--rtl/** | 目录， SoC_lite 的源码 |
| **\| \| \|--soc_lite_top.v** | SoC_lite 的顶层 |
| **\| \| \|--myCPU /** | 自实现 **CPU** 源码 |
| **\| \| \|--CONFREG/** | confreg 模块 |
| **\| \| \|--BRIDGE/** | bridge_1x2 模块 |
| **\| \| \|--xilinx_ip/** | Xilinx IP，包含 clk_pll、 INST_RAM、 data_ram， |
| **\| \|--testbench/** | 仿真文件 |
| **\| \| \|--mycpu_tb.v** | 仿真顶层 |
| **\| \|--run_vivado/** | 目录， 运行 Vivado 工程 |
| **\| \| \|--soc_lite.xdc** | Vivado 工程设计的约束文件 |
| **\| \| \|--mycpu_prj1/** | 目录， Vivado2018.1 创建的 Vivado 工程 1 |
| **\| \| \| \|--mycpu.xpr** | Vivado2018.1 创建的 Vivado 工程， 可直接打开并进行仿真、综合实现且无错 |
| **\| \| \| \|--func.bit** | 89 个功能点测试 bit 文件 |
| **\| \| \| \|--memory.bit** | 运行记忆游戏 bit 文件 |
| **\|-soc_axi_perf/** | 自实现 CPU 的性能测试环境 |
| **\| \|--rtl/** | SoC_lite 的源码 |
| **\| \| \|--soc_lite_top.v** | SoC_lite 的顶层 |
| **\| \| \|--myCPU /** | 自实现 **CPU** 源码。 |
| **\| \| \|--CONFREG/** | confreg 模块 |
| **\| \| \|--ram_wrap/** | 目录， axi ram 的封装层，增加固定延迟设置 |
| **\| \| \|--xilinx_ip/** | 目录， Xilinx IP，包含 clk_pll、 INST_RAM、 data_ram,， |
| **\| \|--testbench/** | 目录， 仿真文件 |
| **\| \| \|--mycpu_tb.v** | 仿真顶层 |
| **\| \|--run_vivado/** | 目录， 运行 Vivado 工程 |
| **\| \| \|--soc_lite.xdc** | Vivado 工程设计的约束文件 |
| **\| \| \|--mycpu_prj1/** | Vivado2018.1 创建的 Vivado 工程 1 |
| **\| \| \|--run_allbench.tcl** | 仿真依次运行 10 个性能测试程序的脚本 |
| **\| \| \| \|--mycpu.xpr** | 性能测试 Vivado 工程 |
| **\| \| \| \|--perf.bit** | 运行性能测试 bit 文件 |
| **\|-soft/** | 功能测试和性能测试软件程序目录 |
| **\| \|--func/** | 89 个功能点测试程序 |
| **\| \|--memory_game/** | 记忆游戏测试程序 |
| **\| \|--perf_func/** | 性能测试程序 |

## （二）设计演示结果

预赛设计部分仅实现比赛要求的指令和 Cache，未实现 TLB，现阶段无法运行操作系统，因而展示内容仅为功能测试上板样张与性能测试仿真、上板结果。功能测试样张如下图：

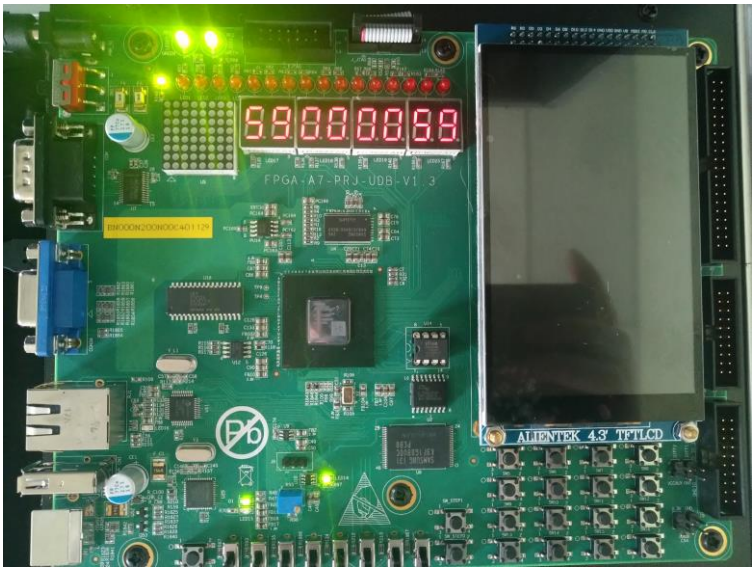


图 8

性能测试统计见下表:

表 15

| 序号 | 测试程序 | myCPU | | | |
|---|---|---|---|---|---|
| | | 仿真计时(16 进制) | 上板计时(16 进制) | | 数码管显示/10：仿真 |
| | | | 数码管显示 | 数码管显示/10 | |
| cpu_clk : timer_clk | | 100MHz : 100MHz | 70MHz : 100MHz | | - |
| 1 | bitcount | 11bd1 | e1339 | 16852 | 1.270 |
| 2 | bubble_sort | 5290f | 46d308 | 7151A | 1.372 |
| 3 | coremark | 14b6a1 | 11f6fcd | 1CBE61 | 1.388 |
| 4 | crc32 | 176f55 | 14c4551 | 213A21 | 1.418 |
| 5 | dhrystone | 27161 | 202301 | 336B3 | 1.316 |
| 6 | quick_sort | 538e3 | 3f0bf6 | 64DFF | 1.207 |
| 7 | select_sort | 314f6 | 281c6d | 402D7 | 1.302 |
| 8 | sha | bf5d2 | a8a852 | 10DDA1 | 1.410 |
| 9 | stream_copy | cdc5 | 4d083 | 7B40 | 0.599 |
| 10 | stringsearch | 75718 | 67b546 | A5EED | 1.413 |

# 四、参考设计说明

## （一）数据通路

数据通路设计参照计算机组成原理课本[3]中所将的流水线数据通路，结合 MIPS 官方推荐的配套书目《DDCA》[4]，根据 MIPS 流水线结构要求的基本器件进行组合，其中流水线冒险、异常处理部分参考书本实现方式进行处理，结合自身对数据通路的分析，完成整个数据通路的设计。

## （二）CP0、除法器

异常处理和特权指令的实现需要实现 CP0，因学校《硬件综合设计》课程已有提供 CP0、除法器的实现，故直接使用。由于课程资源部分直接采纳雷思磊[1]老师的实现成果，包括 CP0 模块、除法器等，故而视为参考设计。其中除法器初始阶段采用学长优化的双边缘触发除法器，仿真阶段可成功实现，但设计不建议使用下降沿触发，因而仍旧采用与 OPENMIPS 相同的除法器。

## （三）Cache

由于性能测试有固定的读写时延，因而需实现 Cache 以优化性能。MIPS 软核设计为哈佛结构，分 RAM 为 INST_RAM 与 DATA_RAM，但 AXI 总线接入的 AXI_RAM 为单一内存模块，前期设计中在 axi_interface 接口模块对取指和访存操作进行仲裁，但设计 Cache 模块时发现，李亚民老师[2]的 Cache 设计除 Cache 模块外，还考虑了哈佛结构到单内存之间的访存仲裁，因此我们将对接 AXI 总线的 axi_interface 模块面向 CPU 的端口改为单端口设计，参照李亚民老师设计的 Cache 及仲裁，由 Cache 流出的数据经过仲裁后，已经成为单端口输出至接口处，由总线流入 Cache 的数据同理。

从设计理念和 Cache 需求而言，都契合预赛 SOC 的接入，且考虑到实现难度，Cache 模块均采用李亚民老师教材的设计。由于李亚民老师仅列举了写直达，块大小为 1 word 的 Cache，因此在不更改外部端口的情况下，我们设计实现写回的 Cache。自实现的 Cache 经测试可以通过功能测试与性能测试仿真，上板验证时出现错误，考虑到得分问题，不得不将 Cache 替换回原始版本，因此性能分数也未做优化。

# 五、参考文献

[1] 雷思磊. 自己动手写 CPU[M]. 北京：电子工业出版社, 2014.
[2] 李亚民. 计算机原理与设计: Verilog HDL 版[M]. 北京：清华大学出版社, 2011.
[3] David A.Patterson, John L.Hennessy. 计算机组成与设计硬件/软件接口[M]. 北京：机械工业出版社, 2007.
[4] 戴维·莫尼·哈里斯, 莎拉 L.哈里斯. 数字设计和计算机体系结构[M]. 北京：机械工业出版社, 2016.